Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 100 (2015) 1696 – 1705

Procedia Engineering

www.elsevier.com/locate/procedia

25th DAAAM International Symposium on Intelligent Manufacturing and Automation, DAAAM 2014

A Study of Cloud/IX Operating System for the ARM-based Data Center Server Platform

Yury Leokhin[a]*, Peter Panfilov[a,b]

[a]National Research University – Higher School of Economics, Myasnitskaya St. 20, Moscow 101000, Russian Federation
[b]Ministry of Economic Development, 1st Tverskaya-Yamskaya St. 1,3, Moscow 125993, Russian Federation

Abstract

Exponential growth in data production and a prominent trend in data center design architecture – a shift from expensive hardware towards a multitude of simple servers – pose new tasks and demand the use of different strategies for data center architects. In this work, a new solutions to distributed systems design are discussed, which are based on Plan9 operating system model. We first overview application and research projects including project of porting Plan9 to the IBM Blue Gene/L supercomputer, project of the Plan9 use in data centers and cloud systems, and projects aimed at distributed embedded systems. Then we introduce a Cloud/IX operating system for the ARM-based server platforms that also follows the Plan9 model and is implemented on top of one of Plan 9 derivatives called 9front - a free software distributed operating system. We also present the experimental testbed setup and results of experimental study of the Cloud/IX on multi-computer server farm in actual data center environment.

Peer-review under responsibility of DAAAM International Vienna

Keywords: operating systems; distributed systems; Plan 9 operating system model; server platforms; data centers; Cloud/IX operating system

1. Introduction

Modern data centers look very different than they did just 10 years ago and it's not just that 10 years is perhaps longer than any computer lifespan. The emergence of Big Data, Cloud Computing, 4G Mobile data, and other modern trends drastically changed the spectrum of industry's tasks and problems. Recent advances in science such

* Corresponding author. Tel.: +7-495-916-88-98; fax: +7-495-917-81-54.
*E-mail address:*yleokhin@hse.ru

Peer-review under responsibility of DAAAM International Vienna
doi:10.1016/j.proeng.2015.01.545

as genetic sequencing and nuclear research made sure we can safely assume that data production and massively parallel processing (MPP) continues its exponential growth. Integration with private and public clouds, server consolidation and full virtualization, more extensive skill set of data center personnel are all consequences of this explosive growth. This poses new tasks and demands the use of different strategies for data center (DC) architects.

An increase in power consumption in DCs constitutes the major problem to the DC market development. Energy-related costs account for about half of the total server maintenance costs of data center ownership, while most of these goes to the provision of power supply and cooling to the servers. In general, we can consider two approaches to solution of the energy efficiency problem, namely: an efficient use of existing facilities and resources (e.g., use of virtualization and cloud computing, allowing to increase utilization rate of available resources and to decrease the equipment needs), and new architectural solutions to the data center designs (e.g., Cisco Unified Computing System).

The energy efficiency problem of the server design is approached at different levels from a processor core, through the single server, and up to the server farm. At processor level, the ARM energy-efficient processor architecture [1] is nowadays considered a solution of choice to the development of server hardware. According to IDC forecasts, by 2015 the ARM architecture can win more than 15% market share of server hardware. In September 2010, the British company ARM Holdings first entered this market with Cortex-A15 processor based on ARMv7-architecture.Their version of the ARM Cortex A15 MPCore, designed for CPU clocks of up to 2.5 GHz, demonstrated the five times performance improvement with respect to the processors used in smart phones while maintaining the same energy consumption levels. Recently, ARM has released the first 64-bit processors - Cortex-A57 and A53, and a new 64-bit architecture - ARMv8, which is nearly ready for mass production.

To leverage the advantages that new energy-efficient processor architecture offers, the server architects request new software solutions both at the operating systems level and at level of the server virtualization layers. An example of developments in this area is the x86 binary code compiler for the ARM architecture designed by the Elbrus Technology [2]. It allows for migration of software written for x86-based servers to the ARM processor architecture. Software emulator will allow unchanged run of applications compiled for the x86-architecture on the ARM server. Recently, ARM have published package additions to the Linux kernel, which provide support to the instruction set of the ARMv8-core. These additions are now implemented in a number of flavors of Linux, including Ubuntu.

Thus, the design and development of an operating system for servers based on ARM processor architecture continues to be a task of great importance, a quality solution to which is expected to allow creation the efficient distributed server farms in terms of performance, power consumption, and scalability.

In this paper, we describe our solution to the distributed systems design problems that we approach with the development of a new operating system called Cloud/IX. The design of our own operating system follows the Plan9 model and is implemented on top of one of Plan 9 derivatives called 9front - a free software distributed operating system [3]. We present the general characteristics of Plan 9-based approach to the design of distributed systems, and introduce the Cloud/IX operating system for data center servers based on ARM processors. Section 2 provides quick discussion of distributed systems application spectrum in its relation to the computer architecture problems. Section 3 presents key features of the Plan 9, accompanied with the examples of Plan 9 use in diverse distributed computing application areas. These include project of porting Plan 9 to the supercomputer platform (for MPP scientific computing applications), project of using Plan 9 in data center server platform (for distributed and cloud systems applications), and a research project of adding real-time scheduling into Plan 9 for distributed embedded systems (DES) applications. Section 4 describes our Cloud/IX operating system for the ARM-based server platforms. The experimental testbed and results of experimental tests of the Cloud/IX are discussed in Section 5. Finally, some conclusions and future work are shown in Section 6.

2. Distributed Applications and its Impact on Computer Architecture Design

There is a huge class of tasks that can be well parallelized, which makes it much easier to run them on multi-node computer architectures – this includes mostly computational tasks, such as fluid dynamics and optical modeling as well as generic data processing, i.e. genetic sequencing and text processing, including web search and indexing. Web search and indexing is, in fact, one of the most, if not the most widespread use of computing resources today.

Some estimates put the percentage of worldwide CPU cycles spent on it as high as 35%, which sounds believable when you count for a fact that a search engine consumes not just its own hardware's resources, but also each and every server's it indexes.

This caused a prominent trend in data center architecture design - a shift from powerful and expensive hardware (like mainframes 25 years ago and HP Superdome about a decade later) towards a multitude of simple servers. These massively parallel architectures can use either Google-like server farms built from off-the-shelf hardware or proprietary blocks forming what today is called a supercomputer (IBM Blue Gene).

2.1. Trends in operating systems research and development

While computer hardware evolved at the blazing speed, the software counterpart obviously could not remain the same. Early operating systems creators didn't care much for architecture – nobody at the time had the experience of writing a program this big. As one of the consequences, those early OS's lacked the modular structure, each and every subroutine could be called globally, and the entire thing was a huge monolithic “blob”. This made scaling and expansion extremely difficult. First OS/360 release took 5 years and 5000 people to write and amassed just over 1 million lines of code. Its successor Mastics, released in 1975, already grew to 20 million lines. It was obvious that without radical review of design principles further advances were impossible.

Thus, modular paradigm was born, and most of the development in modern software engineering is still based on it or its variants. Modular design naturally led to modules with similar functionality grouping together and stratification of OS into hierarchical model. Practically all modern OS's can be subdivided into following levels:

- Hardware support
- Machine-dependent code
- Common kernel mechanisms
- Resource manager
- System calls API
- Utilities.

Sometimes levels are split or combined, like in nanokernel and microkernel architectures, sometimes even swapped (exokernel), but the basic structure remains more or less the same.

Another huge step in OS design was made when IBM introduced its Virtual Machine that abstracted the underlying hardware from the lowest level of OS. This made possible system partitioning and running several instances of OS on the single physical machine. For a couple of decades, virtualization was exclusively mainframe feature, but it was, of course, bound to propagate into server and workstation world. Nowadays, all major processor manufacturers include hardware virtualization support (VT for x86, TrustZone for ARM) [4].

It is estimated that today over 50 percent of all server workloads are virtualized and this figure is projected to reach 86 percent by 2016 [5]. Virtualization also spread to workstations where it is widely used as a low-cost software alternative to acquiring dedicated hardware for test and debugging purposes. Lately, it got even to the mobile and embedded segments of the market, where its benefits are security, interoperability and, once again saving on hardware – a virtual multimedia processor can be as good as a dedicated physical one [6].

Virtualization allows computational resource sharing and partitioning, but there is also need for exactly the opposite – not slicing the existing system into a number of virtual machines, but uniting the resources of multiple systems into a bigger and more powerful “supersystem”.

While virtualization techniques are nowadays ubiquitous, including hardware manufacturers' support (VT for x86, TrustZone for ARM) and well known software solutions (VMWare servers and stations, Oracle VBox, Xen), aggregation is a much more complex problem. Simply speaking, if a virtual node, from software viewpoint, is indistinguishable from a physical system, the topology of an aggregated system is quite different from a single node. And, of course, effective use of these aggregated resources requires some sophisticated techniques.

2.2. Modern parallel computing infrastructures for distributed systems

Perhaps all distributed tasks can be put into two large groups – transactional processing and supercomputing. In transactional processing there's a need for servicing huge amount of data and/or transactions while supercomputing involves mainly computational activity. In other words, one is heavy on I/O, the other – on CPU cycles. These days, most transactional processing is done on Linux clusters running either MapReduce or some of its variants. Web search applications are primarily responsible for emergence of parallel computing infrastructures like Google's MapReduce [6] and Microsoft Dryad [7].

On the other hand, supercomputers (or, rather, MPP computers) traditionally include two flavors of nodes – I/O nodes interfacing the outside world and running Linux and compute nodes with almost always proprietary and tightly crafted for performance kernels that carefully minimize expensive events like context switching and cache thrashing. IBM Compute Node Kernel (CNK) used on Blue Gene family of IBM supercomputers is one such example.

Linux got into its dominant position in industry because of huge body of code – almost each and every application has been ported there, and, also not in the last, because of support from industry giants like IBM, HP and Cisco. Other noted players include FreeBSD/OpenBSD/NetBSD family of systems, QNX, VxWorks and, of course Microsoft Windows.

3. Related works: Plan9 operating system revisited

Nowadays, there is a renewed interest in another OS – Plan9 and its derivatives. Plan9 is an OS developed in the late 1980s and early 1990s at AT&T Bell Laboratories by the group of researchers and engineers that included some of the original UNIX creators [8]. In Plan9 design they attempted to straighten out what they thought went wrong with UNIX and its ancestors. When introduced to the USENIX community in 1992, it was received very well, with reviews ranging from carefully optimistic to outright ecstatic branding it “a UNIX killer”.

Killing UNIX did not happen – we can only guess for specific reasons, but the general consensus seems to be that while Plan 9 was in many ways superior to UNIX, it just failed to gain critical mass on the improvements [9]. Simply speaking, UNIX and later Linux as one of UNIX flavors were not as elegant but still good enough. This, combined with its massive code base, put it in an industry leader position.

Plan9, meanwhile, found a niche as hobbyist and research system. It has, as any great but underachieving project would, a small but dedicated army of followers. Its impeccable pedigree and elegant design also make it very attractive as a subject in Operating Systems courses in academia.

Plan9 is based on three major principles:

- All resources are named and represented by files in a filesystem
- There is a standard protocol, 9P, for accessing files across node boundaries
- Separate filesystems can be joined into a single private name space

It was aggressive application of these principles that kept Plan9 consistently compact and robust through the years and a major rework in 2000-2004.

Some of Plan9 features turned out to be so attractive they were adopted by mainstream UNIXs. Most prominent of those is, perhaps, a filesystem interface to system per-process statistics - /proc filesystem. Linux's /sys filesystem representing system-wide resources is another nod in that direction. Plan9 also introduced UTF-8, a full and honest n^2 set of native and cross-compilers and linkers for all supported architectures, and some other nifty innovations.

One of the most attractive Plan9 qualities is its compact size. Historically, it was introduced “when things were small” and even Linux was not the monster we know today. And it managed to stay that through the years. For example, cat utility resident footprint on Ubuntu 12.04 is 384K while its Plan9 counterpart is just 11K. Similarly, most standard utilities common for both systems show a factor of 10 to 30 in memory footprint. Cache usage is, of course, much more conservative in Plan9 as well, which it even more important for performance.

This 'tight and robust' paradigm made Plan9 an attractive candidate for embedded systems design. There it was always, although marginally, present, particularly in network equipment and storage systems. The distributed

processing model of Plan 9 is very effective and flexible, and it is attractive for embedded systems. The 9P protocol is useful for inter-system communication. The private name space of Plan 9 also enables flexible and safe distributed processing in embedded systems. Plan 9 can run on various hardware platforms and is highly suited to building large distributed systems. A typical Plan 9 installation would comprise one or more file servers, some CPU servers and a large number of terminals (user workstations). The small size and straightforward structure of (most of) its source code, and low system management overhead, makes it particularly suitable for distributed embedded systems (DES) applications. The server world, though, lusts for Plan 9's other features – and first of all a relatively thin 9P protocol and the ease of inter-node communication by manipulating name spaces. This leads to a higher level of abstraction – applications are agnostic of their execution details. They can run anywhere on any node in the system, on any architecture. Client can run, within one session, several programs on geographically separated machines. This improves modularity of any project by representing any information or data as a set of plain files [10].

9P protocol was implemented for several foreign systems, including Linux. Actually, for Linux there are is a 9P server allowing accessing files on a Linux server from Plan9 station and 9P client to access Plan9 files from Linux computer. This is very handy for cross-platform development.

As proof of the renewed interest in Plan9 OS we'll take a look on three projects.

3.1. Plan9 OS on Supercomputer Platform

The first project was done jointly by IBM, Bell Labs and Sandia National Labs and involved porting Plan 9 on IBM Blue Gene/L supercomputer [11]. After it was benchmarked [12], the initial results were quite discouraging – on a compute node it was performing up to 30 times slower on SAXPY runs than dedicated Compute Node Kernel. Then, after it was attributed to high number of page faults because of the smaller page size in Plan 9 (4KB against CNK's 1MB) and a custom version of Plan 9 with 1MB page size was used, it performed on par and even slightly better than CNK (see Table 1).

Table 1. Relative performance of CNK and Plan9 on IBM Blue Gene/L.

Stride	CNK	Plam9	Plan9 (*1MB page*)
1	3.7539700e-01	3.8000000e-01	3.7000000e-01
101	3.7721600e-01	1.6400000e+00	3.8000000e-01
201	3.7926300e-01	3.4000000e+00	3.8000000e-01
301	3.8119100e-01	4.9100000e+00	3.8000000e-01
401	3.8263600e-01	6.3300000e+00	3.8000000e-01
501	3.8444400e-01	7.7600000e+00	3.9000000e-01
601	3.8661300e-01	7.7700000e+00	3.9000000e-01
701	3.8841900e-01	7.7600000e+00	3.9000000e-01
801	3.8986500e-01	7.7600000e+00	3.9000000e-01
901	3.9227500e-01	7.7700000e+00	3.9000000e-01
1001	3.9444300e-01	1.1750000e+01	4.0000000e-01

This, in fact, encouraged the teams involved in these projects to proceed to design Plan 9 based supercomputer operating system called NIX [13, 14].

3.2. Plan9 OS on Server Platform

Another recent project is Clive [15] targeted for use in distributed and cloud systems. Clive architecture is based on Plan9 and NIX. Clive is written in Go and is distributed under MIT license. It is of particular interest because it is:

- One of the designs of a distributed system around a CSP-like style with applications and components using CSP channels for communication – networks, pipes and other I/O
- An interesting attempt to eliminate software stack in clouds, running applications written in Go without any OS, on top of a simple hypervisor or even bare hardware.

The core of the project is a modified Go compiler and runtime, adopting network interface support and operation without the underlying OS. In effect, a compact and robust nature of OS kernel based on Plan9 allows for each and every application to have its own OS kernel.

The choice of Go as system language is based on its parallel nature with channel transactions. This naturally suits distributed programming and solves network latencies problems. Clive is built around "zx" - universal resource access protocol based on Plan9's 9P with similar resource naming and namespace manipulation semantics.

3.3. Plan9 OS in Embedded and Real Time Systems

Plan 9 is a highly-portable operating system designed for a networked world. It has been run on systems ranging from small embedded devices to supercomputers. While Plan 9 on IBM Blue Gene/L, NIX and Clive are all research projects, we can also recall successful commercial use of Plan 9. This includes now phased out NIX and LocalDirector projects at Cisco and several ongoing storage products by Coraid. Plan 9 was also used in several embedded environments. For instance, it is the system inside the *Viaduct*, a computer system that provides an encrypted bridge between Lucent employees' home computers and the corporate intranet [16].

In many embedded systems, some applications have stringent real-time requirements. Traditionally, general-purpose operating systems have never been good at guaranteeing deadlines. Various attempts have been made to introduce real-time schedulers to general-purpose operating systems. Plan 9 is a general purpose operating system that is finding increasing use as an embedded systems platform. Recently, researchers at Bell Labs have started using Plan 9 as a basis for a wireless base station, which requires a hard real-time system. Their experience with the design, implementation, integration and use of a hard-real-time scheduler in their operating system Plan 9 as an embedded operating system can be found in [17].

Another example of Plan 9-based operating systems projects for embedded systems is an LP49 operating system [18]. This is a research OS that intend to combine best features of Plan 9 OS with best features of L4 microkernel. On the one hand, Plan 9 is a smart component-oriented OS: File servers, etc., are organized as normal processes. Kernel devices (e.g. devxxx) are also component-oriented. If the OS functions can be extended merely by adding components, an embedded system developer can always use the optimum OS. In addition, the system servers in Plan 9 are user mode processes; hence, Plan 9 enables easy program development. On the other hand, L4 microkernel is a small micro-kernel that provides very efficient threads and has a very efficient and flexible IPC function. Researchers have adapted the L4 microkernel and are building the Plan 9-based OS modules on L4. A micro-kernel OS has many processes, and they run with intercommunication. Hence, the IPC's performance dominates the performance of the OS. The L4 microkernel's high-speed IPC enables them to build a high-performance OS, and Plan 9's smart facilities attain flexible distributed processing. LP49 will implement the 9P protocol, so it can communicate with Plan 9. The assumed applications of LP49 are embedded systems. Such applications use various kinds of devices, and hence, many kind of device drivers must be developed.

4. Cloud/IX operating system – a Plan9-based solution to ARM-based server platforms

When selecting the basic operating environment for the development of Cloud/IX OS we used multiple criteria, including among others the support for distributed operations, scalability, license purity, easy porting of device drivers and applications, easy deployment and support, standard interfaces, minimal system services overhead.

The main advantage of the 9front system for distributed server application is its ixP protocol, which allows for managing local and distributed resources by simple mappings onto the namespace.

Perhaps the most notable disadvantage of 9front is the difference between its set of system interfaces and the POSIX, which is a traditional standard solution for the similar products.

Here we can consider two different approaches to solution of this problem. First, an APE (ANSI / POSIX Environment) package was developed for 9front - the best approximate of the system interfaces to the POSIX. Second, our team in association with AltLinux company undertake efforts of porting Linux on the ARM and microTCA-based server platform (ARM, microTCA). This will permit easy adaptation to the target platform of many applications developed for the Linux, including traditional cluster applications.

It should be noted that many useful features of 9front design were adapted to Linux. This applies, in particular, to the ixP protocol, the use of which in Linux is now possible at the level of mounted file systems that allows for exchange of files between Linux and 9front.

Since our Cloud/IX is based on 9front, it inherits all the features of its prototype. The system is based on three principles:

- Resources are named and are available as files in a hierarchical file system
- ixP standard protocol for access to local and remote resources
- unbounded hierarchies provided by diverse services are linked together into an own hierarchical file namespace.

ixP protocol implements multiple transactions, each of which sends a request from the client process to the local or remote server and returns the result. ixP controls the file system, not just files. Access to the files occurs at the byte level, not blocks, which distinguishes ixP from protocols such as NFS and RFS [19].

At present, a β-version of the Cloud/IX operating system is developed, and a work is performed on porting the most popular and commonly used software applications (e.g., nginx – a web-server and a mail proxy-server running on Unix-like operating systems).

5. Experiments with Cloud/IX

We have carried out tests of the software prototype of the ARM-based server platform in order to study the stability of the ported nginx http-server on heterogeneous Cloud/IX system and its scalability. The tests were performed in the Data Center at the Systems and Solutions Ltd. on a distributed computer system that comprises 24 x86-based computers (blade servers), organized into 3 system racks each with 8 blade servers. All blade servers are equipped with at least one Ethernet 1000Mbps controller.

5.1. Experimental Setup

During the experimental testing, we have monitored the load level in cluster nodes with OS services (separately for each subsystem), which allows to conclude about potential ways of performance improvement. To display the results of the monitoring, a special purpose software was developed to collect statistics from multiple nodes in a cluster, to aggregate it on the single node, and to transform monitoring results into format suitable for the analysis and display in real-time. The software receives data about the node's and network interface's workloads and displays it in a visual form on a web page. The solution is implemented using the following technology stack:

- Server part of application is written in Clojure – a lisp-dialect implementation for the JVM and libraries: Ring, Compojure, Webbit, Clj-json
- Client part is implemented in ClojureScript – a dialect of Clojure that translates into a regular JavaScript, executed in the browser
- Implementation of message passing mechanism from the server to the client is based on WebSockets technology
- Dynamic rendering of the graphical elements is realized by working with Canvas element (HTML5 specification).

To generate requests to the load balancer nginx the httpperf utility is used. Httpperf measures the performance of web server and provides a flexible environment for generating workloads for the HTTP-server and for measuring its performance. At the end of test run, it generates a report, which contains three sections: general results, a group

dedicated to compounds and group. In the load generating mode, it generates requests with the substitution of the growing numbers, which digits are used as components of the path to the resource.

Scenario of testing the effectiveness of load balancing require the creation of a nginx file hierarchy one each node, so that their paths with respect to the root directory of nginx match the query, formed by httpperf, and thus the total amount of data would exceed the size of RAM in each node. Under these conditions, the disk subsystem becomes the bottleneck at each node, so that it becomes possible to evaluate the effect of workload parallelization.

The same sequence of non-recurring requests is submitted to the balancer and to the separate node, and the results are compared. Performance of individual components and of the entire cluster is also tested by repeated (identical) queries. Httpperf allows you to adjust the number of requests per unit of time, which is reflected in the number of requests processed in parallel. The test is carried out separately for downloading large files and for downloading small files, allowing you to identify the various potential bottlenecks in the ported nginx. In this test scenario, the entire contents of the file in the cache of the operating system, and a disk subsystem is no longer a bottleneck.

We wanted the test results to reflect the performance of the server solutions (nginx), as opposed to the entire client-server complex. For this, it requires either a presence of multiple client computers, generating queries simultaneously, or the use of a system for running the client, which outperforms significantly a set of all nodes in the cluster (without disk subsystem that is not used by the client actively). For this study, we have chosen the second solution.

5.2. Experimental results

The main measure of the test results is the number of processed requests per second for the final statistics httpperf. Another indicator, which is of great importance in the content analysis of the workloads and bottlenecks - the requests rate (number of requests per unit of time) at which the server stops successfully handle all connections. To obtain this a several runs of httpperf are carried out, where the option *-rate* is consistently increased, accompanied by a statistics check of successful requests. Monitoring of the cluster nodes load with operating system's services (separately for each subsystem) during exercise testing also allows for making conclusions on potential performance improvement measures (Figure 1).

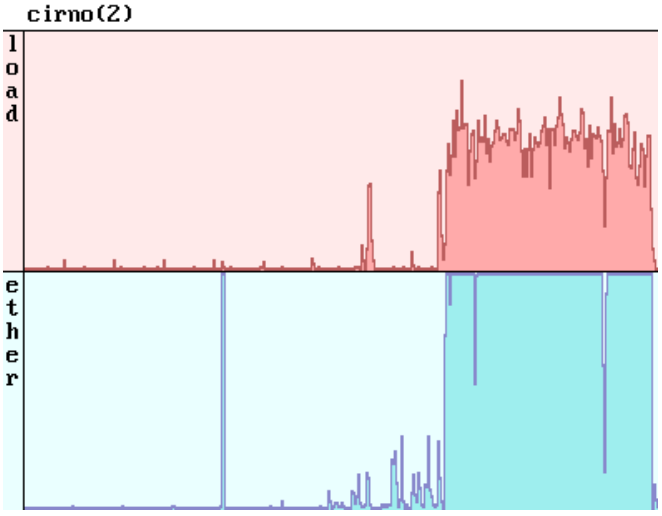

Fig. 1. Load levels of the CPU and network adapter of the key node during testing.

Four series of experiments were conducted during experimental testing. In each, a cluster of blades of a size from 4 to 24 was used (Table 2). The table shows the scalability of nginx server, running under the Cloud/IX operating system in Data Center.

Table 2. Results of tests of Cloud/IX.

Test series	Connections per sec., average	Test time, sec., average	Response rate, min., conn./sec.	Response rate, mod.., conn./sec.	Response rate, max., conn./sec.	CKO, %
1	128,0	1022.6	74,4	128,0	196,8	7.4
2	324,0	404.9	177,6	321,6	477,6	9.8
3	416,0	315,3	258,2	414,4	592,0	11,4
4	1054,6	125,9	750,0	1051,6	1591,2	16,8

However, in the course of the experiments, we have revealed some problems associated with the connections between nginx servers running under the Cloud/IX. When servers were overloaded with requests, nginx discarded packets (connection refused) that arrived during the processing phase of another package's connection. A plot of the fraction of failures for a series 2 of experiments with respect to the request rate is shown in Figure2. We believe that this is due to incorrect processing of sockets in the Cloud/IX, as well as due to peculiarities of the TCP-stack implementation in the 9front distribution, used as a basis for our own development. After examining the way 9front refers to the TCP protocol, we have found that before the SYN + ACK the 9front kernel does not check whether the client has enough space in the processing queue, and discards RST after the connection is established, which is consistent with RFC793. RFC793 does not describe how the server should handle the overflow in a queue: before or after the ACK. Thus, the above-mentioned problem related to the work with the TCP stack can be treated as a peculiarity of Cloud/IX, which should be taken into account in further work with software that uses the TCP protocol.

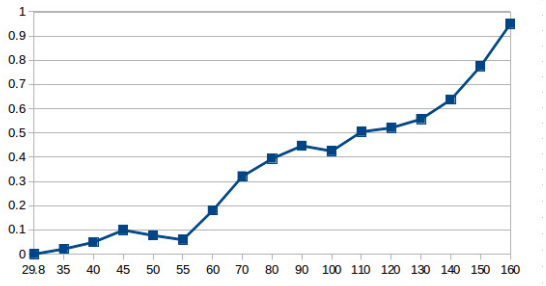

Fig. 2. Dependence of proportion of refusals on request rate, Comm./100 ms.

In addition, this Cloud/IX operating system was not originally made for the use of sockets, and for their implementation ported BSD-sockets were taken, which do not take into account the architectural features of the network stack of the Cloud/IX. Based on the results of tests we assume that the network performance can be significantly improved by using sockets, which take into account these features. In the future, implementation of the BSD-socket protocol on top of 9p, which is used in the Cloud/IX.

Unlike many other operating systems, Cloud/IX from the very beginning is not providing a special programmed interfaces to access devices. Instead, the system is supposed to control the interactions through the file system, which may be accessed through the 9p protocol, including devices. The interaction is intended to be realized by conventional I/O operations. This fact explains the absence of sockets in the Cloud/IX and thus the developers are challenged with their implementation under the operating system.

Following the results of experiments in a testbed, it has been found out that the networked server applications can be ported to and used in the Cloud/IX system and demonstrate performance comparable to similar devices. In addition, a number of features of the 9front-core were identified and developers are challenged to find ways to work with them and their description.

6. Conclusion

In this paper, we introduced the distributed systems design approach, which is based on Plan 9 operating system model. We showed that underlying principles of Plan 9 OS are well suited to capture the distributed processing mechanisms arising from parallel and distributed computational/programming models using supercomputer, server platforms, and distributed embedded systems examples. The definition of filesystem interface together with the ease of inter-node communication by manipulating file name spaces enables for generation of applications regardless the peculiarities of their underlying computer system hardware. They can run anywhere on any node in the system, on any architecture. This results in application including the full functionality captured by the model as well as the communication needs imposed by the allocation of workloads onto a distributed computing system. The 9P protocol included in Plan 9 implementation provides the basis for building scalable distributed systems architectures and a clue for distribution of tasks to the available system nodes. This improves modularity of any project by representing any information or data as a set of plain files. Additionally, the system servers in Plan 9 are user mode processes; hence, Plan 9 enables easy program development.

To sum it up, we can clearly see the trend of recently renewed interest in Plan 9 OS and its derivatives. There are several ongoing projects focusing on real-time and MPP support extensions to Plan 9.

Acknowledgements

This work was funded by the project “Development and organization of high technology production of energy efficient multiprocessor hardware-software server complexes for government and corporate information systems and data centers”, which has been carried out at the National Research University Higher School of Economics (HSE).

References

[1] S. Orlov. Revolution ARM. Journal of network solutions. LAN №11, 2012. Available at: http://www.osp.ru/lan/2012/11/13032394/.
[2] Startup Elbrus Technologies’ emulator will allow ARM processors to work with x86-applications. Available at: http://servernews.ru/596643.
[3] "Plan 9 from the People's Front of cat-v.org (9front)", *NineTimes*, June 17, 2011, retrieved September 13, 2012.
[4] T. Laplante, Virtualization has surpassed 50 percent of all server workloads, DataCenterPost.com, March 20, 2014. Available at: http://datacenterpost.com/2014/03/virtualization-surpassed-50-percent-server-workloads.html.
[5] O. Kharif, Virtualization goes mobile, Bloomberg Businessweek. Technology, April 22, 2008. Available at: http://www.businessweek.com/stories/2008-04-22/virtualization-goes-mobilebusinessweek-business-news-stock-market-and-financial-advice.
[6] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, 6th Symposium on Operating Systems Design & Implementation (OSDI’04), December 6-8, 2004, USENIX 2004, pp.137-149. [PDF].
[7] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks, European Conference on Computer Systems (EuroSys’07), Lisboa, Portugal, March 21-13, 2007. [PDF].
[8] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, Plan 9 from Bell Labs, *Computing Systems,* vol. 8, no. 3, 1995, pp. 221–225. Available at: http://plan9.bell-labs.com/sys/doc/9.html.
[9] E.S. Raymond, The Art of Unix Programming, Thyrsus Enterprises, 2003.
[10] D. Presotto, P. Winterbottom, The Organization of Networks in Plan 9. Available at: http://plan9.bell-labs.com/sys/doc/net/net.html.
[11] E. Van Hensbergen, C. Forsyth, J. McKie, and R. Minnich, Petascale Plan 9 on Blue Gene, USENIX 2007 Annual Technical Conference (USENIX ATC’07), June 17-22, 2007, Poster Session. [Abstract].
[12] R.G. Minnich, J. Floren, and A. Nyrhinen, Measuring kernel throughput on Blue Gene/P with the Plan 9 research operating system, in: Proceedings of the 6th International Workshop on Plan 9 (IWP9), Athens, GA, USA, October 12, 2009. [PDF].
[13] J. McKie, J. Floren, Edging Towards Exascale with NIX. [PDF]
[14] NIX is a new multicore OS based on Plan9. Available at: http://code.google.com/p/nix-os/.
[15] F.J. Ballesteros, CSP-style Network, File, and System Services in Clive. Lsub Systems Lab, Universidad Rey Juan Carlos, Madrid, TR Draft, May 23, 2014. [PDF].
[16] S.J. Mullender, P.G. Jansen, Real Time in a Real Operating System, in: Herbert, Andrew James (et al.) (Eds.), Computer Systems. Theory, Technology, and Applications, Springer, 2004, pp. 213-221. ISBN 978-0-387-21821-2. [PDF].
[17] S.J. Mullender, J. McKie, Real Time in Plan 9, in: Proceedings of the 1st International Workshop on Plan 9 (IWP9), December 4-5, 2006, Madrid, Spain. [PDF].
[18] Y. Sato, K. Maruyama, LP49: Embedded system OS based on L4 and Plan 9, in: Proceedings of the 4th International Workshop on Plan 9 and Inferno (IWP9), Athens, GA, USA, February 21-23, 2009. [PDF].
[19] H. Trikey, APE - The ANSI/POSIX Environment, Plan 9 Programmer's Manual, Volume 2, AT&T Bell Laboratories, Murray Hill, NJ, 1991.